REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE Professional Paper	3. DATES COVERED

4. TITLE AND SUBTITLE The ACETEF HLA Interface	5a. CONTRACT NUMBER
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Dr. Stephen O'Day, John McMaster	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Air Warfare Center Aircraft Division 22347 Cedar Point Road, Unit #6 Patuxent River, Maryland 20670-1161	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Naval Air Systems Command 47123 Buse Road Unit IPT Patuxent River, Maryland 20670-1547	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The Air Combat Environment Test and Evaluation Facility (ACETEF) at Patuxent River Naval Air Station received a certificate of High Level Architecture (HLA) compliance in September 1999. The software vehicle for attaining compliance for the ACETEF simulation objective model (SOM) was the ACETEF FHL SWEG/JIMM interface. This interface is written in a flexible architecture which permits straightforward federation object model (FOM) extensions to the SOM object classes and the addition of FOM specific methods to the SOM base classes. The FOM specific extensions can be archived apart from the interface and re-used as needed. This interface is currently being used in the Joint Test and Training Capability Assessment (JTTCA) and the Joint Strike Fighter (JSF) Virtual Strike Warfare Environment 7 (VSWE7) exercises. A description of the software architecture and the philosophy behind the architecture is described. Methodologies for FOM specific code development are discussed.

15. SUBJECT TERMS

Air Combat Environment Test and Evaluation Facility (ACETEF) Simulated Warfare Environment Generator (SWEG)
Joint Interim Mission Model (JIMM) Simulation object model (SOM)

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON Dr. Stephen O'Day / John McMaster
a. REPORT Unclassified	b. ABSTRACT Unclassified	c. THIS PAGE Unclassified	Unclassified	5	19b. TELEPHONE NUMBER (include area code) (301) 342-6101 / 342-6357

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39-18

20040107 128

The ACETEF HLA Interface

Dr. Stephen C. O'Day
J.F. Taylor, Incorporated
21610 South Essex Drive
Lexington Park, MD 20653

John W. McMaster
NAWCAD
Air Combat Environment Test&Evaluation Facility
Building 2109
48150 Shaw Road
Patuxent River, MD 20670

Keywords:
flexible FOM, FOM re-use

ABSTRACT:*The Air Combat Environment Test and Evaluation Facility(ACETEF) at Patuxent River Naval Air Station received a certificate of High Level Architecture (HLA) compliance in September 1999. The software vehicle for attaining compliance for the ACETEF simulation object model (SOM) was the ACETEF HLA SWEG/JIMM interface. This interface is written in a flexible architecture which permits straightforward federation object model (FOM) extensions to the SOM object classes and the addition of FOM specific methods to the SOM base classes. The FOM specific extensions can be archived apart from the interface and re-used as needed. This interface is currently being used in the Joint Test and Training Capability Assessment(JTTCA) and the Joint Strike Fighter (JSF) Virtual Strike Warfare Environment 7 (VSWE7) exercises. A description of the software architecture and the philosophy behind the architecture is described. Methodologies for FOM specific code development are discussed.*

1. Introduction

The ACETEF HLA Interface was designed to provide a convenient, highly configurable interface to the SWEG/JIMM shared memory(SWEDAT) data structures, protocol, and any ACETEF laboratory asset interfaced to that environment.

The Simulated Warfare Environment Generator(SWEG) is the forerunner of the Joint Interim Mission Model(JIMM) which combines the features of the Navy's SWEG model with those of the Air Force SUPPRESSOR model. These models are all language-driven and permit the developer to create objects that have capabilities, susceptibilities, resources, tactics and systems(i.e. sensors, weapons, communicators, disruptors, thinkers and movers). Object information is stored in shared memory while interactions between objects appear in asset action messages.

The interface includes implementations of the eight basic object classes and seventeen basic interaction classes of the ACETEF SOM which is published in the DMSO SOM library[1]. The SOM Object hierarchy is based on "distributable" characteristics of simulation entities represented. Any FOM classes may be implemented by extending the SOM classes to include attribute data and interaction objects which match the federation format.

Those classes which may be extended are included in the user FOM files. The standard SOM files are not intended for modification since the goal is to produce an archivable, retrievable FOM implementation that will compile and link with the fixed portion of the interface code.

2. The ACETEF SOM

A brief description of the ACETEF SOM classes follows.

2.1 SOM Object Classes

The eight basic object classes consist of the following: Simulation_Entity, Moving_Entity, Live_Vehicle, Site, Abstract, Adhoc_Weapon, Simulation_Environment and Remote_Environment.

Simulation_Entity is a class which contains pointers to SWEG/JIMM platforms but not systems. The Moving_Entity class is derived from Simulation_Entity and instances of this class move. Instances of the Adhoc_Weapon class enter the game after the start and have limited capabilities.

Live_Vehicle and Abstract are derived from Moving_Entity. Live_Vehicle instances contain pointers to all types of SWEG/JIMM systems whereas Abstract instances have only disruptor and weapon system pointers. Site is derived directly from Simulation_Entity and has pointers to all types of SWEG/JIMM systems.

Simulation_Environment is a class which contains general scenario location and calendar date information. Remote_Environment is a class which contains connectivity and federate_id information.

2.2 SOM Interaction Classes

The seventeen types of interactions(whose names make them fairly self-explanatory) are as follows:

detonates_warhead
fires_a_weapon_at
declares_hit_or_miss
changes_sensor_status
changes_disruptor_status
changes_communicator_status
changes_sensor_mode
aborts_inflight_shot
removes_from_scenario
changes_sensor_pointing
changes_jammer_pointing
sensor_chance
sensor_occulting_changes
changes_signature
tracking_status_update
perception_update
explicit_communication

3. RTI Services Supported

This interface release supports the following RTI services[2]:

createFederationExecution
destroyFederationExecution
joinFederationExecution
resignFederationExecution
publishObjectClass
publishInteractionClass
subscribeObjectClassAttributes
subscribeInteractionClass
startRegistrationForObjectClass
stopRegistrationForObjectClass
registerObjectInstance
discoverObjectInstance
updateAttributeValues
reflectAttributeValues
sendInteraction
receiveInteraction
deleteObjectInstance
removeObjectInstance

requestAttributeValueUpdate
provideAttributeValueUpdate
negotiatedAttributeOwnershipDivestiture
requestAttributeOwnershipAssumption
attributeOwnershipDivestitureNotification
attributeOwnershipAcquisitionNotification
attributeOwnershipAcquisition
requestAttributeOwnershipRelease
attributeOwnershipReleaseResponse
cancelNegotiatedAttributeOwnershipDivestiture
cancelAttributeOwnershipAcquisition
confirmAttributeOwnershipAcq.Cancellation
isAttributeOwnedByFederate

3.1 Service Call Constraints

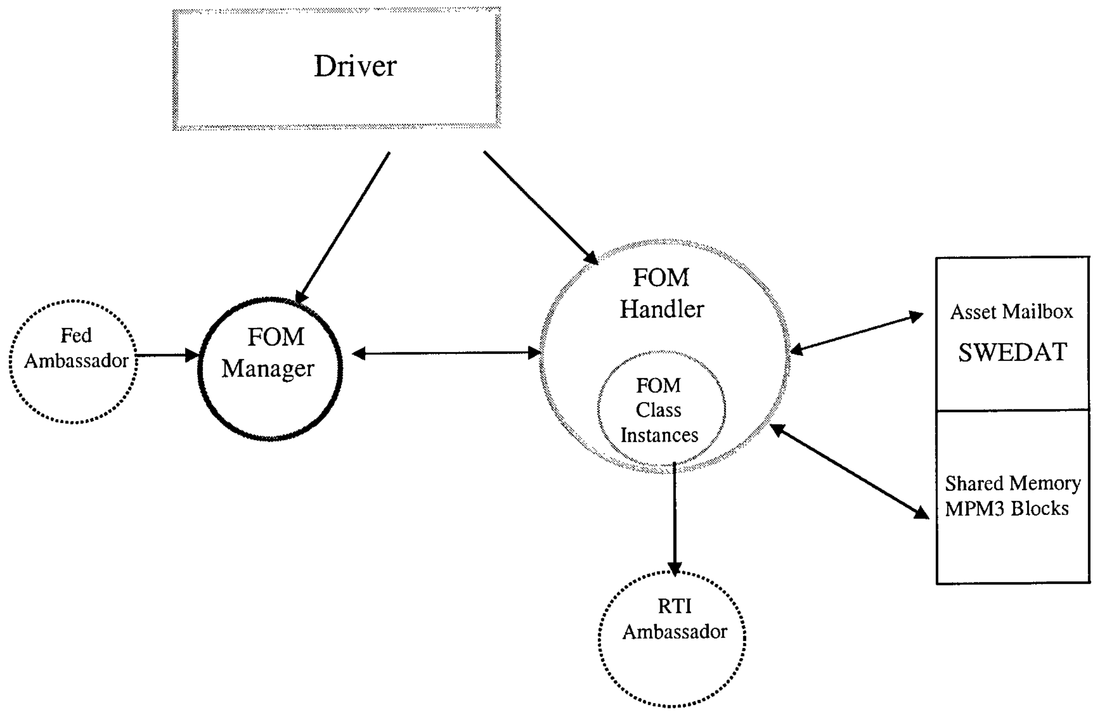
Clearly not all services will be employed in any one federation so the FOM designers must consider the following issues:

A.) Which Federate will create the federation executive?
B.) Will attribute updates be (in HLA jargon) "pushed" (updateAttributeValues, reflectAttributeValues) or "pulled" (requestAttributeValueUpdate, provideAttributeValueUpdate)? A Federate can opt to have some object classes use the "push" method, and some use the "pull" method.
C.) Will object registration be automatic or controlled by the startRegistrationForObjectClass / stopRegistrationForObjectClass services?
D.) If attribute transfer will be used, will it be done via the "push" or the "pull" services (negotiated or non-negotiated).

The interface is designed to implement a federate within a federation which accepts the following rules:
A.) The Federate is neither time constrained nor regulating, but externally time-synchronized.
B.) Attribute transfer may occur at the start of a game, after registration, or immediately after an object enters the game, but at no other time.
C.) Attributes of a registered object instance may be transferred only once during the object's existence.
D.) All objects that will play in the federation must be known and identified by unique registration object tags for the duration of the scenario.

4. Interface Design Overview

The following diagram is a structural overview of the interface:



4.1 Interface Structure

The Driver file contains the C++ main function which has a while-loop for periodically "ticking" and making service calls to the RTI ambassador using Fom_handler methods. The Driver file invokes methods of the Fom_handler and Fom_manager classes through single instances of each class constructed in "main". Two posix threads are launched by the Driver as well.

The Driver's Fom_handler instance contains instances of all the interface objects and instances of all the objects which hold incoming and outgoing interaction data. The Fom_handler also holds pointers to shared memory and to Fom_manager linked lists.

The Fom_manager instance is responsible for mapping a FOM into a specific SWEDAT-resident scenario (run under SWEG/JIMM or other engine), and for performing standard housekeeping and initialization sequences (create/join, read roster, attach to scenario, resign/destroy federation). Callbacks from the federation ambassador are routed to the Fom_handler through the Fom_manager.

The Fom_handler class instance contained in the Driver also has methods which are invoked by the two posix threads launched by the Driver. The "mailbox" thread periodically checks the mailbox from SWEG to the interface. It responds to asset action dispatches by writing data into the appropriate FOM objects and/or flagging interactions to be sent by the main loop thread.

The "SWEDAT" thread periodically cycles through all the FOM objects, checking last update times and copying new information to or from SWEDAT as the necessity for update is indicated. The Fom_handler plays the key role in the functions just described because it contains the objects which interface between the federation and SWEG/JIMM players.

The configuration setup(CSET) file is a roster of federation players used by the Fom_manager to generate a linked list of interface objects. The Fom_handler then dynamically allocates an array of interface objects which contain pointers to both SWEDAT and the Fom_manager linked lists.Sometimes federations insist on publication at the platform and system/device level. A properly written roster will result in a one to one correspondence between federation and interface objects with appropriate pointers to SWEDAT.

4.2 Interface Execution

As execution begins, the Fom_manager reads the CSET file and builds a linked list of "player-nodes" using the CSET information. After locating SWEDAT, the interface pre-installs the FOM objects using the information on the "player-nodes". This creates a set of FOM objects, managed by the Fom_handler, some of which are attached to corresponding objects in SWEDAT, and others that are awaiting their later appearance in the federation.

The Fom_manager then creates and/or joins the federation and begins publication/subscription and registration. If attribute transfers are to take place concerning objects that are in the federation from start, those attribute transfers take place next.

Finally, the interface notifies SWEG/JIMM of its readiness, the scenario starts, and the interface main loop is entered. At this point both posix threads are running. The main loop ticks the RTI, processes callbacks and then calls the FOM handler method trigger_objects to allow each object a chance to use RTI services.

When a user presses CTL-C, a signal handler setup in hla_if.c attempts a graceful resignation from the federation, displaying a message indicating that action and telling the user to press CTL-C a second time if the federate appears to be hung. A second CTL-C will force an exit.

5. FOM Implementation

To implement a FOM using the ACETEF HLA interface, the developer must decide which of the ACETEF SOM classes most resemble the classes of the FOM. The SOM classes are then modified to accommodate the style of interaction and attribute configuration required to play in a particular federation.

The CSET roster allows one to specify an "object root" for each named FOM class and you may have several FOM classes derived from a SOM class. On the SWEDAT side of the interface, there may be multiple federation objects each of which points to only a part of a SWEG/JIMM player(since players are composed of platforms,elements and systems).

It is also possible to include interactions which do not occur in the SOM but which are required to play in the federation. The long term goal is to then add these to the SOM if it is felt that more general future applications may arise from these federation-specific interactions.

The described approach has been successfully employed by ACETEF for the JTTCA project[4] and in an integration test for VSWE7[5].

5.1 File Structure

There are certain files designated for federation specific modification and certain files for which modification is discouraged from the standpoint of archivability and configuration management.

All of the base class definitions are contained in the hla_standard.h file. The hla_if_manager.cc and basic_handler.cc files contain the hla_if_manager and basic_handler base class methods, respectively. The acetefFederationAmbassador methods are located in the ambassadors.cc file. SOM object and interaction methods are found in acetef_som.cc. Created player queue functions are located in create_queue.cc and fifo.cc. All of these files are considered to be standard SOM files and are not to be modified for federation-specific applications.

The acetef_fom.h file contains the derived FOM object class definitions. The derived and added FOM interaction class definitions are contained in fom_interactions.h. The fom_HandleGroup.h possesses the derived FOM handle classes. All of the FOM object, interaction and handle class methods appear in acetef_fom.cc. The complex data types which support the FOM classes are found in cdt.h. The derived Fom_manager class definition and methods may be found in fom_manager.h and fom_manager.cc. The fom_handler.h file contains the Fom_handler class definition and fom_handler.cc contains the methods.

6. The Federation CSET Roster

This interface is designed to support federations that are "fully specified", meaning, all possible instances of all possible object classes must be predetermined and enumerated prior to federation. This interface does not support "surprise" instances in a federation. If a variable number of instances are possible, depending upon the events that occur during a federation's execution, a range of numeric identifiers must be assigned to the total set of possible instances, prior to execution. For instance, you federate may wish to play a Surface-to-Air Missile (SAM) battery, and you will publish the classes "sam-battery" and "sam-missile". Any "sam-site" instance may fire up to 6 "sam-missile" objects during the federation, depending upon the availability of air targets. Prior to federation, you must decide upon a numeric "Class ID" for the "sam-site" and "sam-missile" classes, and a unique, numeric "Instance ID" for each instance of each class. Since each "sam-site" can fire up to 6 "sam-missiles", you will have to define a range of six unique numeric values for each of those possible "sam-missiles". This is done so that the interface can build its CSET roster of objects.

Upon startup, the interface reads the CSET file and pre-instantiates all the FOM object instances mentioned in it. These FOM objects are the go-betweens between the Federation and SWEDAT. There are four types of FOM objects:
A.) Remotely owned and present in the game from the start.
B.) Locally owned and present in the game from the start.
C.) Remotely owned, appears in the game sometime after the start.
D.) Locally owned, appears in the game sometime after the start.

Note that these categories are irrespective of attribute transfers; "ownership" refers to the federate owning them at their creation.

The CSET file can be created by hand or can be generated using the GUI utilities. Its purpose is to define the "roster" of federation objects for the interface. It maps the federation class id number and federation instance id number to the SWEDAT player semantic code and global instance id. Note that the federation id numbers are federation-wide codes for all classes and instances in the federation that must be agreed upon by federation participants – these are distinct from the class and object handles provided by the RTI at runtime.

The CSET also enables one to run a given FOM with various SWEG scenarios, by changing the semantic codes and global id's to match those of the new scenario. The new scenario entities must have the appropriate capabilities to match the FOM requirements.

When the interface is started, seven arguments must be provided on the command line:
•[shmem/scram] SWEDAT location, local shared memory or SCRAMNET.
• [asset ID]Numeric identifier of asset to be interfaced.
• [CSET filename]CSET filename.
• [fedex name]Federation executive name.
• [FED filename]Name of FED file to use (OMDT output file).
• [join flag]If 0, simply join the federation, if 1, create the federation, then join.
• [registration flag]If 0, register all objects at start, if 1, wait use start/stop registration callbacks.

Note that the asset id is found in the SWEG/JIMM scenario CDB asset declarations.

7. Summary

The ACETEF HLA interface supports the object and interaction classes of the ACETEF SOM. The interface is comprised of a multi-threaded architecture which interacts with the RTI and with SWEG/JIMM shared memory. The CSET roster file reduces the burden on the RTI and on other federates by allowing the interface to read in from file federation information which is pre-determined rather than requesting this information through attribute updates. The existing interface has been thoroughly tested and is available through the Joint Interim Mission Model office web-site and from ACETEF. The full SOM implementation is expected to be released later this year.

8. References

[1] DMSO Simulation Object Model(SOM) library http://oml.msiac.dmso.mil
[2] Simulated Warfare Environment Generator User's Guide Vol.4 Interfaces, ACETEF,1999.
[3] RTI 1.3 Programmer's Guide,DMSO,1999.
[4] Joint Test and Training Capability Assessment Interface Control Document, M.Payne Project Director,1999.
[5] Hough,M. Major General USMC: "JSF- The Affordable Solution",http://www.jast.mil

Author Biographies

STEPHEN O'DAY is a Senior Scientist at ACETEF(Patuxent River Naval Air Station). He works in the Model Interfaces group and has taken the lead in the HLA interface development effort for the JIMM and SWEG models. Dr. O'Day has implemented the IEEE DRM4 and DRM8 in SWEG and JIMM. Dr. O'Day received his Ph.D. in physics from the University of Maryland in 1990 for work done at Fermi National Accelerator Laboratory in Chicago,IL.

JOHN MCMASTER is the leader of the Model Interfaces group at ACETEF. John designed the software architecture of the ACETEF HLA interface. He also designed and developed the DISI interface. John has participated in numerous distributed simulation events at the Atlantic Ranges and Facilities during his career. Among these are the JSF VSWE events, ACETEF-REDCAP integration, JADS-EW, HLA Engineering Protofederation(1996), STOWE-1994 and KERNEL BLITZ. John received his BSEE from Drexel University in 1982 and is a member of IEEE and NMIA.